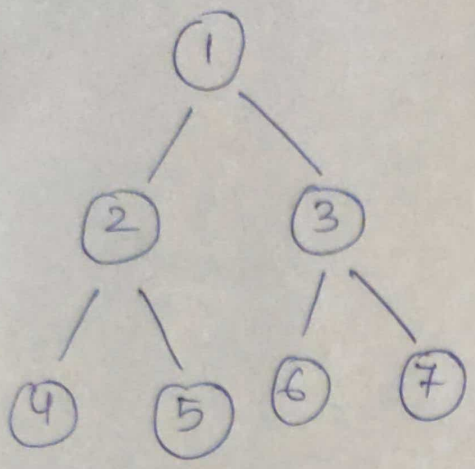
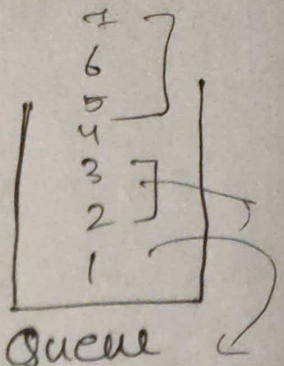
level order Traversal :-

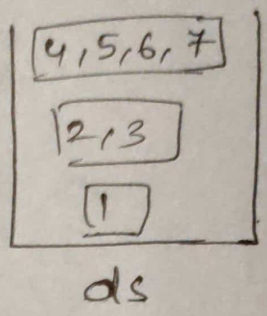


1

2 3

4 5 6 7



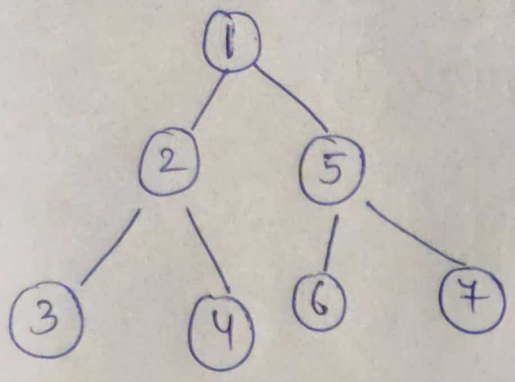


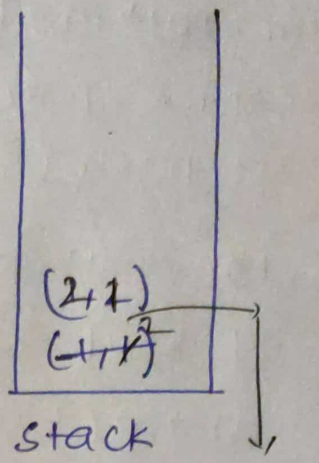
Binary Tree :-

Code :-

```
vector<vector<int>> levorder(TreeNode *root){
    vector<vector<int>> ans;
    queue<TreeNode *> q;
    q.push(root);
    while(!q.empty()){
        int size = q.size();
        vector<int> level;
        for(int i=0;i<size;i++){
            TreeNode *node = q.front();
            q.pop();
            if(node->left!=NULL) q.push(node->
        if(node->right!=NULL) q.push(node->
        level.push_back(node->val);
    }
    ans.push_back(level);
  }
  return ans;
}
```

Preorder Postorder & Inorder in One Traversal:-





Preorder → 1 2 3 4 5 6 7

Inorder → 3 2 4 1 6 5 7

Postorder → 3 4 2 6 7 5 1.

On 1st iteration (num=1),

preorder → (1,1) becomes

↓ will have 1. (1,2) → [check if there exist left of 1]

On next iteration take value (2,1) if num=1

preorder → (2,2)

↙ check left of 2 repeat same

[1,2]

(3,2) → inorder.

(3,3) → postorder.

```
if (num==1)
  preorder
  ++
  left.
if (num==2)
  inorder
  ++
  right
if (num==3)
  postorder
  ++
```

In this way the process repeats, if (left/right) exist then num resets to 1 else it increases.

Code:-

```
vector<int> preInPostTraversal(TreeNode *root) {
    stack<pair<TreeNode *, int>> st;
    st.push({root, 1});

    vector<int> pre, in, post;
    if(root == NULL) return;
    while(!st.empty()) {
        auto it = st.top();
        st.pop();
```

```
        if (it.second == 1){
            pre.push_back(it.first->val);
            it.second++;
            st.push(it);
            if(it.first->left != NULL) {
                st.push({it.first->left, 1});
            }
```

part of pre inc 1 to 2
pushing left side of the tree.

```
        else if (it.second == 2) {
            in.push_back(it.first->val);
            it.second++;
            st.push(it);

            if(it.first->right != NULL) {
                st.push({it.first->right, 1});
            }
        }
```

pre inc 2 to 3
right

---

u dont push

```
        else {
            post.push_back(it.first->val);
        }
    }
}
```